Московский институт электроники и математики имени А. Н. Тихонова

Кафедра информационной безопасности киберфизических систем

Москва 2025

# Лекция 3: Основы работы веб-приложений

Курс: Технологии пентестинга

Автор: Космачев Алексей Алексеевич

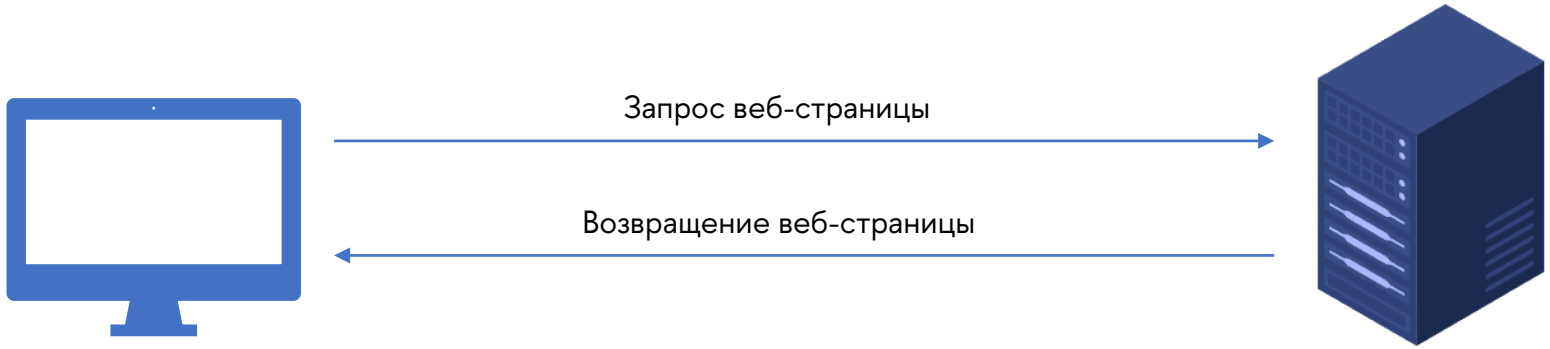
# План лекции

1. Клиент и сервер
2. Приложение, веб-сервер и веб-сайт
3. HTTP
4. Ключевые заголовки
5. Прокси-сервер
6. Архитектуры веб-приложений
7. Intercept Proxy


1. Разведка
2. Сбор информации
3. Поиск уязвимостей
4. «Пробив» периметра
5. Закрепление
6. Повышение привилегий
7. Внутренняя разведка
8. Боковое перемещение
9. Реализация критического события

# Клиент и сервер

# Клиент-серверная архитектура




Клиент

Потребитель информации

- Веб-браузер
- Консольное приложение
- ПО
- И т.д.

Сервер

Производитель информации

# Клиент-серверная архитектура

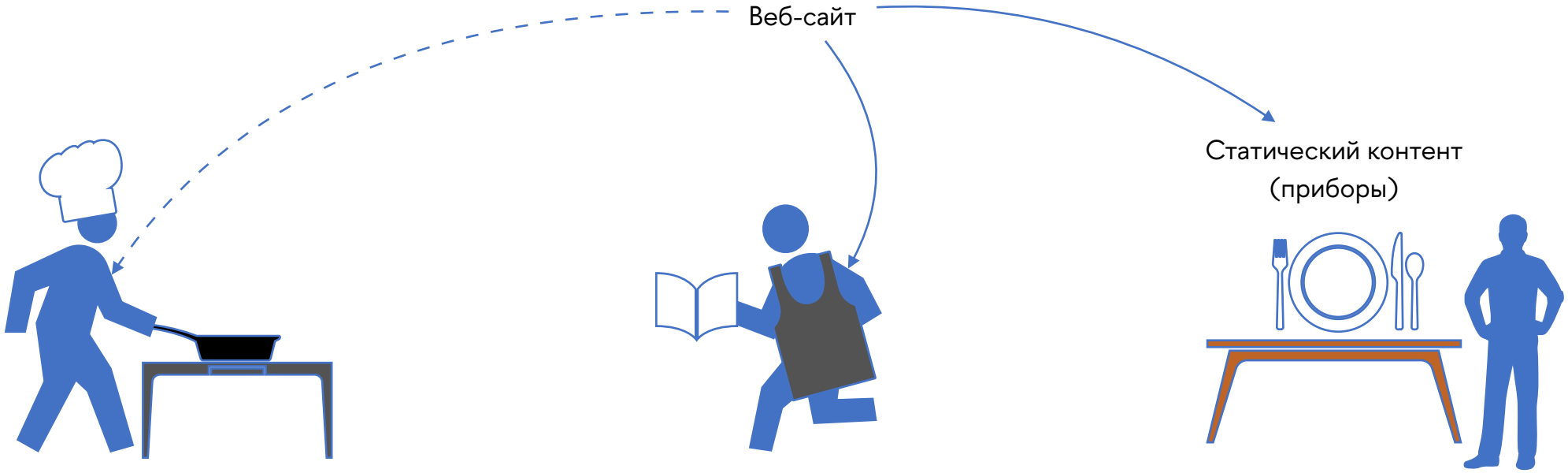
# Какой минимальный набор нужен, чтобы запустить веб-сайт?

# Какой минимальный набор нужен, чтобы запустить веб-сайт?

- Веб-сервер и статический контент

# Приложение, веб-сервер и веб-сайт

# Различия




Приложение (повар)

- Реализует логику (язык программирования)
- Обрабатывает данные
- Взаимодействует с окружением (БД)
- Производит динамический контент

Веб-сервер (официант)

- Обрабатывает запрос клиента
- Понимает куда конкретно нужно доставить запрос в приложении
- Доставляет статический контент

Клиент

# Различия


Приложение (повар)

- Реализует логику (язык программирования)
- Обрабатывает данные
- Взаимодействует с окружением (БД)
- Производит динамический контент

Веб-сервер (официант)

- Обрабатывает запрос клиента
- Понимает куда конкретно нужно доставить запрос в приложении
- Доставляет статический контент

Клиент

*Веб-приложение без статического контента - API или генерирует статический контент на лету

## Веб-сервера

- Nginx
- Apache HTTP Server
- Microsoft IIS
- Caddy
- LiteSpeed
- Tomcat
- Gunicorn / uWSGI
- ...

## Популярные связки

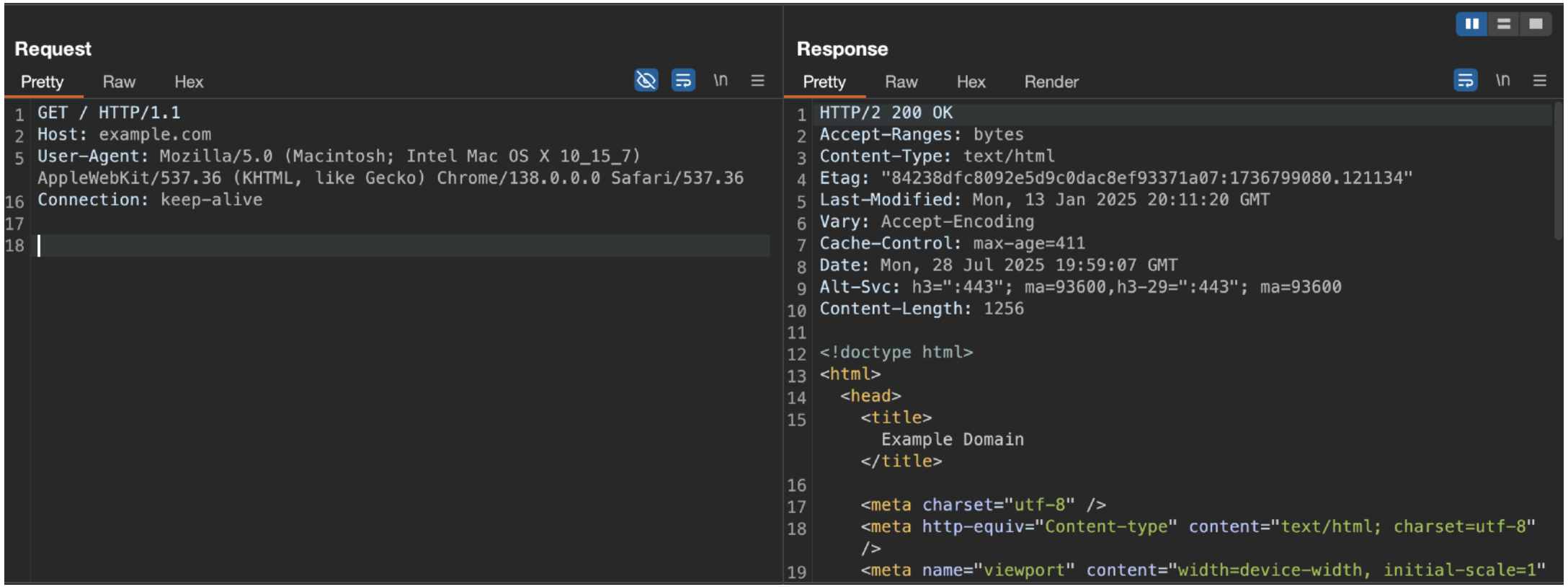
- Nginx + Python (Django/Flask/FastAPI)
- Nginx + Node.js (Express/NestJS)
- Apache + PHP (WordPress/Laravel)
- IIS + C# (ASP.NET Core)
- Tomcat + Java (Spring Boot)
- Caddy + Go (Gin/Echo)

# HTTP

# HTTP

- Протокол прикладного уровня
- Работает по принципу запрос-ответ
- Бывает шифрованным (HTTPS) и нет (HTTP)
- Имеет различные версии

# Запрос-ответ



**Request**

Pretty Raw Hex

```
1  GET / HTTP/1.1
2  Host: example.com
5  User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
   AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36
16 Connection: keep-alive
17
18
```

**Response**

Pretty Raw Hex Render

```
1  HTTP/2 200 OK
2  Accept-Ranges: bytes
3  Content-Type: text/html
4  Etag: "84238dfc8092e5d9c0dac8ef93371a07:1736799080.121134"
5  Last-Modified: Mon, 13 Jan 2025 20:11:20 GMT
6  Vary: Accept-Encoding
7  Cache-Control: max-age=411
8  Date: Mon, 28 Jul 2025 19:59:07 GMT
9  Alt-Svc: h3=":443"; ma=93600,h3-29=":443"; ma=93600
10 Content-Length: 1256
11
12 <!doctype html>
13 <html>
14   <head>
15     <title>
         Example Domain
       </title>
16
17     <meta charset="utf-8" />
18     <meta http-equiv="Content-type" content="text/html; charset=utf-8"
       />
19     <meta name="viewport" content="width=device-width, initial-scale=1"
```

# Какой минимальный набор HTTP-запроса?

# Минимальный запрос

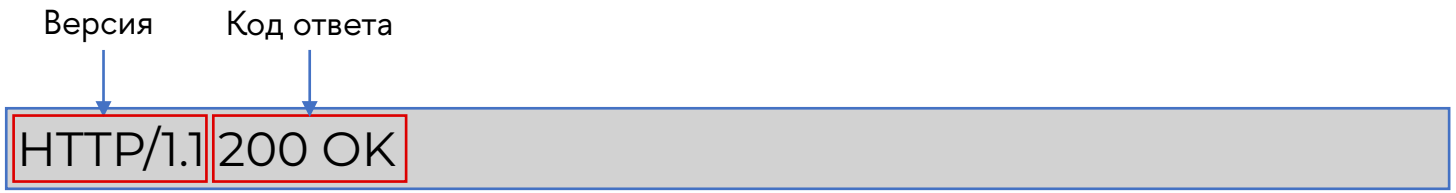
```
GET / HTTP/1.1
Host: example.com
```

# Составные части

# Минимальный ответ

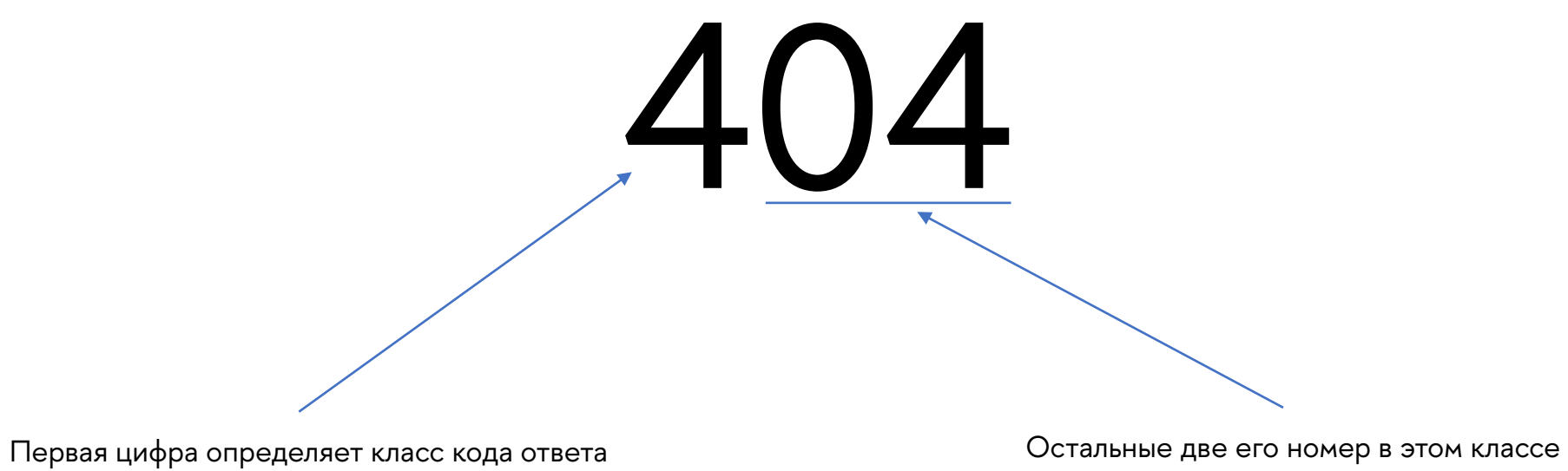
```
HTTP/1.1 200 OK
```

# Составные части

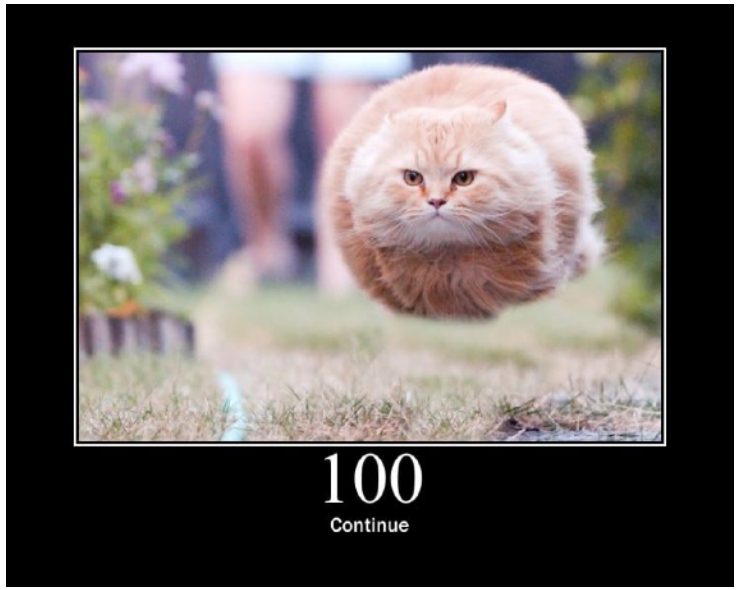


# Коды ответа

- Три цифры



# Классы кодов ответа






1xx - информационные, например:

- 100 - Continue
- 101 - Switching Protocols

2xx - коды успеха, например:

- 200 - OK
- 201 - Created

3xx - перенаправления, например:

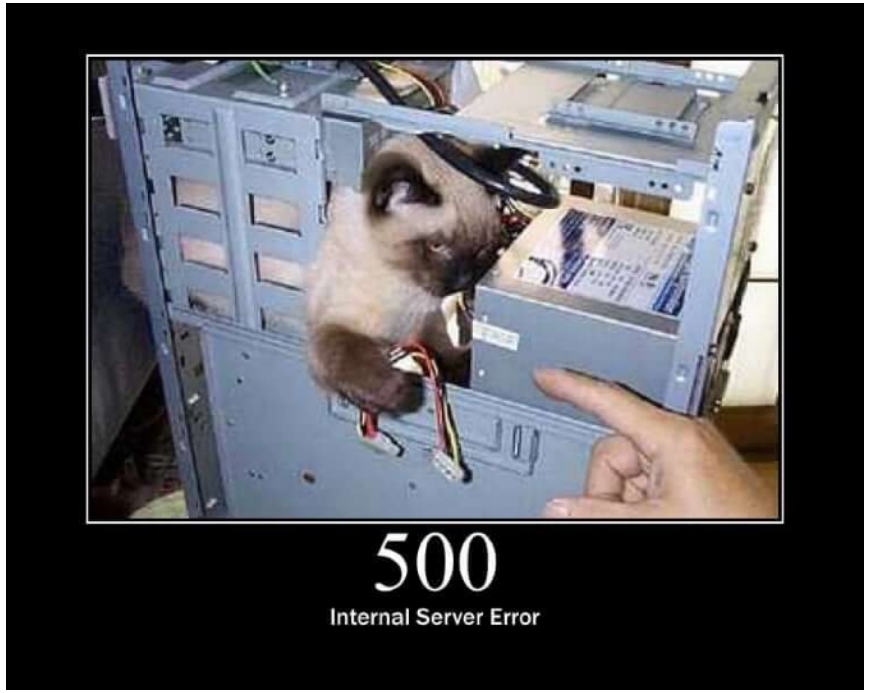
- 301 - Moved Permanently
- 302 - Found

https://http.cat/

# Классы кодов ответа


4xx - ошибка на стороне клиента, например:

- 401 - Unauthorized
- 403 - Forbidden

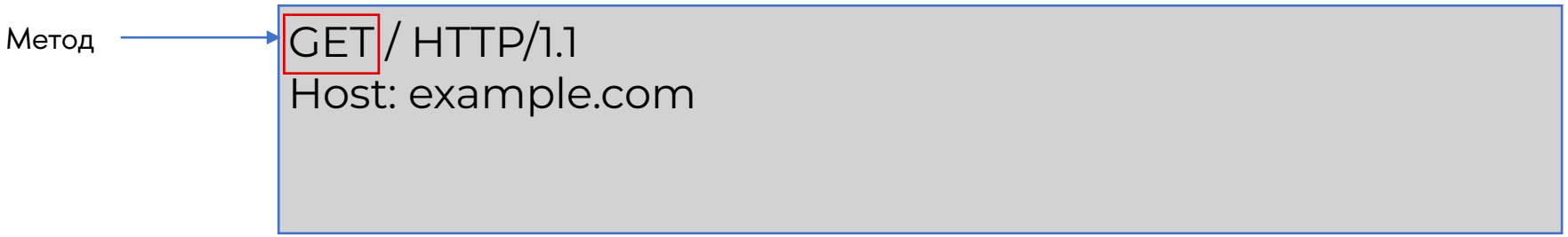



5xx - ошибка на стороне сервера, например:

- 500 - Internal Server Error
- 504 - Gateway Timeout

https://http.cat/

# Метод




- Определяет тип производимого действия

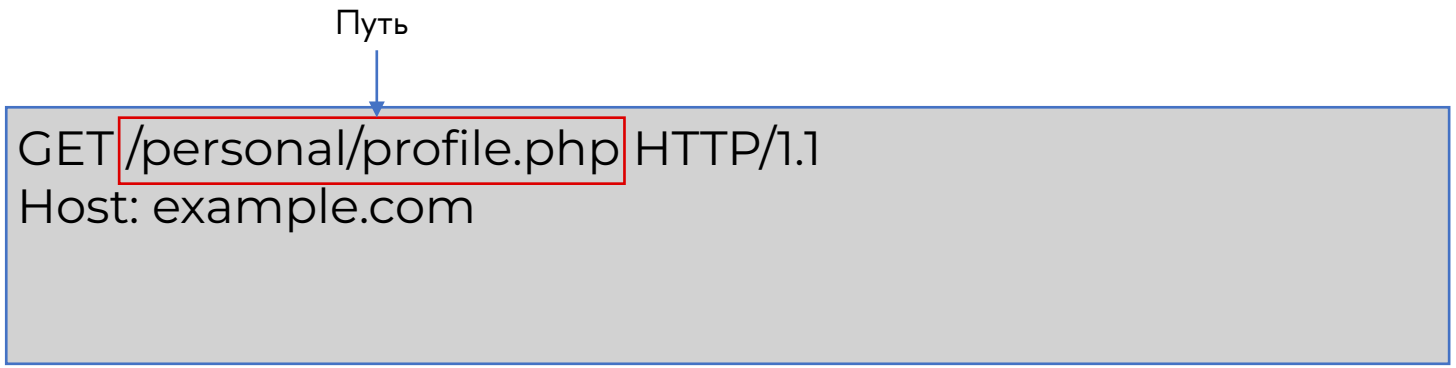
# Основные HTTP-методы

**Properties of request methods**

| Request method | RFC | Request has payload body | Response has payload body | Safe | Idempotent | Cacheable |
|---|---|---|---|---|---|---|
| GET | RFC 9110 | Optional | Yes | Yes | Yes | Yes |
| HEAD | RFC 9110 | Optional | No | Yes | Yes | Yes |
| POST | RFC 9110 | Yes | Yes | No | No | Yes |
| PUT | RFC 9110 | Yes | Yes | No | Yes | No |
| DELETE | RFC 9110 | Optional | Yes | No | Yes | No |
| CONNECT | RFC 9110 | Optional | Yes | No | No | No |
| OPTIONS | RFC 9110 | Optional | Yes | Yes | Yes | No |
| TRACE | RFC 9110 | No | Yes | Yes | Yes | No |
| PATCH | RFC 5789 | Yes | Yes | No | No | No |

https://en.wikipedia.org/wiki/HTTP

# Путь




- Позволяет веб-серверу понять, какой именно фрагмент кода приложения требуется запустить (HTTP Routing)

# HTTP Routing Types

**File system Routing**

1. Фиксируется web root  веб-сервера (/var/www/html)
2. Файлы приложения помещаются физически на файловую систему (/var/www/html/profile.php)
3. При обращении по пути веб-приложения происходит адрессация к конкретному файлу (/profile.php -> /var/www/html/profile.php)

**Annotation-Based Routing (Decorator Routing)**

- Пути декларируются в декораторах соответствующих классов/функций

```
@Controller('users')
export class UsersController {
  @Get(':id')
  getUser(@Param('id') id: string) { ... }
}
```

# HTTP Routing Types

**Code-Based/Imperative Routing**

- Пути декларируются напрямую в коде

```
const app = express();
app.get('/users/:id', (req, res) => { ... });
```

**Servlet Mappings (Java Servlet API)**

- Имеется специальный xml-файл, который определяет соответствие путей обработчикам (Servlet)

```
<!-- web.xml -->
<servlet-mapping>
   <servlet-name>UserServlet</servlet-name>
   <url-pattern>/users/*</url-pattern>
</servlet-mapping>
```

# HTTP Routing Types

**Configuration-File Routing**

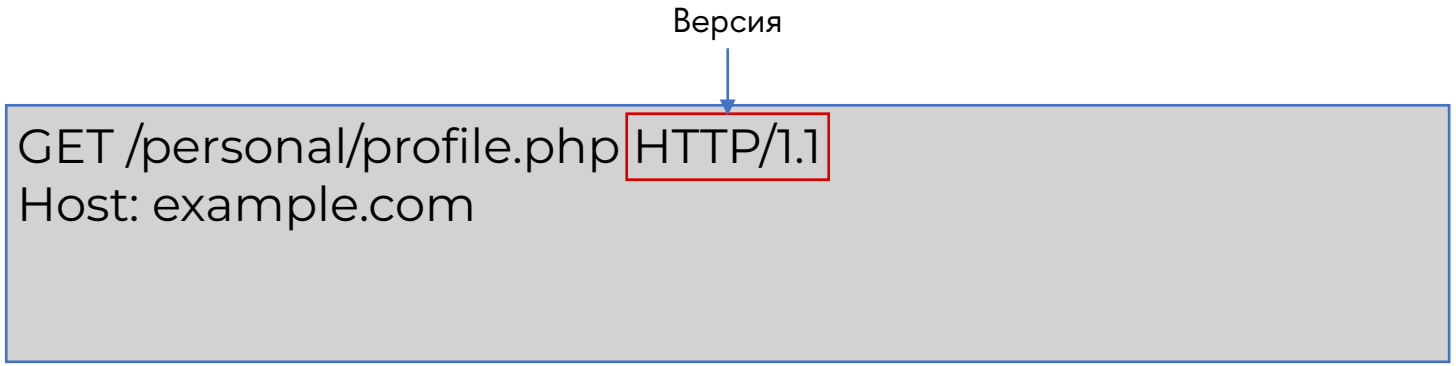
- Имеется файл-конфигурация, определяющий соответствие пути и класса/функции

```
# config/routes.yaml
user_profile:
  path: /users/{id}
  controller: App\Controller\UserController::show
```

**Convention-Based Routing**

- Пути наследуются исходя из названий (/users/show/:id запустит метод show класса UsersController)

# Версия




- Определяет формат, в котором были отправлены данные

# Версии HTTP

## HTTP/0.9

- Совсем древний, нигде не используется
- Нет заголовков, только метод GET, только HTML
- Проблема: слишком примитивный

## HTTP/1.0

- Древний, но где-то еще встречается
- Появились методы HEAD, POST
- Появились заголовки
- Появились типы контента (не только HTML)
- Появились коды состояния (Response codes)
- Проблема: одно соединение на объект

## HTTP/1.1

- Старенький, но до сих пор самый популярный
- Появилась возможность не обрывать запрос (Connection:Keep-Alive)
- Обязательный заголовок Host
- Появилось кеширование, сжатие, чанкованные данные (Transfer-Encoding:chunked)
- Новые коды состояния и методы (PUT, DELETE, TRACE,...)
- Проблема: Головная Блокировка (Head-of-Line Blocking - HOL): Медленный ответ на первый запрос в конвейере блокирует все последующие в очереди

# Версии HTTP

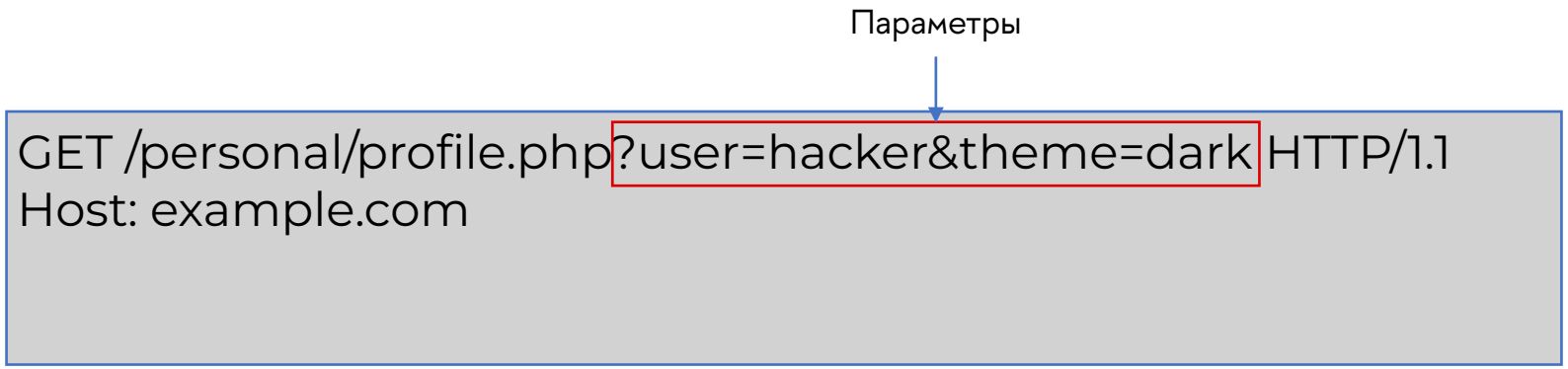
## HTTP/2

- Широко используется, но пока не победил первую версию
- Бинарный протокол (бинарные фреймы вместо текста)
- Мультиплексирование (параллельная отправка запросов в рамках одного TCP-соединения)
- Приоритизация запросов (напр. JS важнее картинок)
- Сжатие заголовков
- Возможность отправить объекты до того как их запросили
- Проблема: HOL Blocking уровня TCP: Потеря одного TCP-пакета блокирует всю поток данных в соединении, так как TCP требует строгой последовательности доставки.

## HTTP/3

- Активно внедряется, но еще развивается
- Использует QUIC вместо TCP (работает поверх UDP)
- Встроенное шифрование
- Очень быстрый
- Умеет быстро переключаться между сетями
- Проблема: Требует поддержки и на уровне клиента, и на уровне сервера

# Параметры




- Позволяют пользователю передать какую-то информацию приложению

# Способы передачи параметров

**Обычный GET-запрос**

```
GET /personal/profile.php?user=hacker&theme=dark HTTP/1.1
Host: example.com
```

**GET-запрос RESTful-сервиса**

```
GET /personal/profile.php/hacker/dark HTTP/1.1
Host: example.com
```

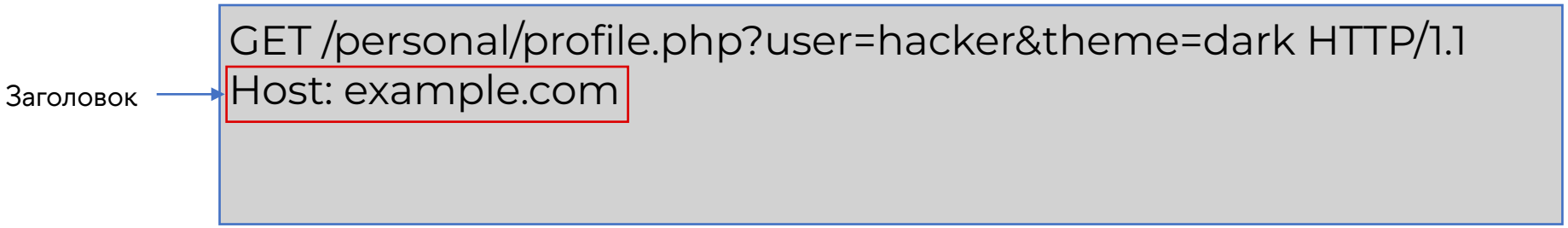
# Способы передачи параметров

**POST-запрос**

```
POST /personal/profile.php? HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 22

user=hacker&theme=dark
```

# Заголовки




- Передают специальную информацию/метаданные приложению/веб-серверу

# Ключевые заголовки

# Заголовки авторизации и места хранения подобных данных в браузере

**Cookies**

- Имеют меньший размер (~4КБ)
- Отправляются автоматически при каждом запросе
- Доступны из JS, но это можно ограничить флагами безопасности
- Имеют 3 флага безопасности: HTTPOnly, Secure, SameSite
- Время жизни можно регулировать
- Отправляются при помощи заголовка **Cookie:**

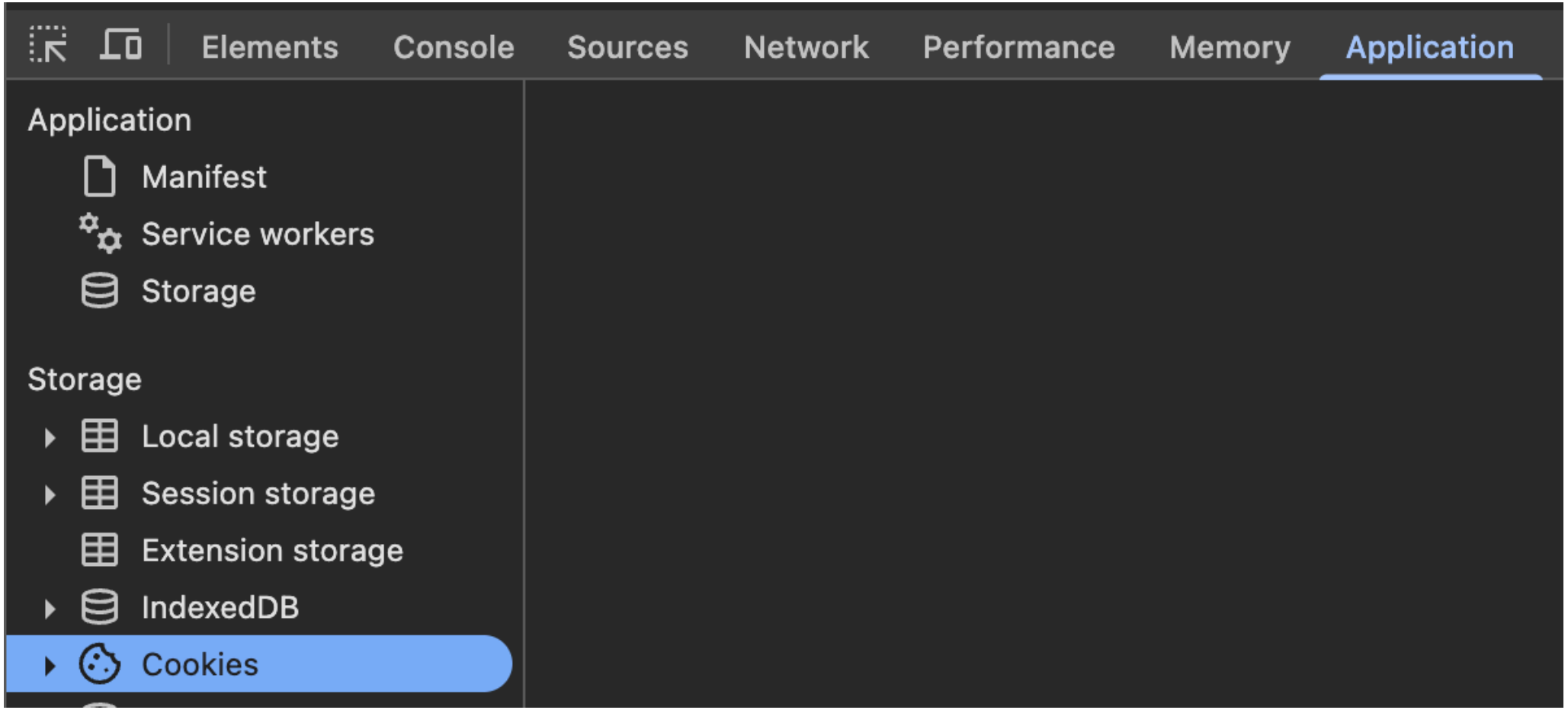
**LocalStorage**

- Имеют больший размер (5-10МБ)
- Никогда не отправляются автоматически
- Всегда доступны из JS
- Нет лимита времени жизни
- Отправляются при помощи заголовка **Authorization:**

**SessionStorage**

- Имеют больший размер (5-10МБ)
- Никогда не отправляются автоматически
- Всегда доступны из JS
- Живут пока активна вкладка браузера
- Отправляются при помощи заголовка **Authorization:**

# Заголовки авторизации и места хранения подобных данных в браузере

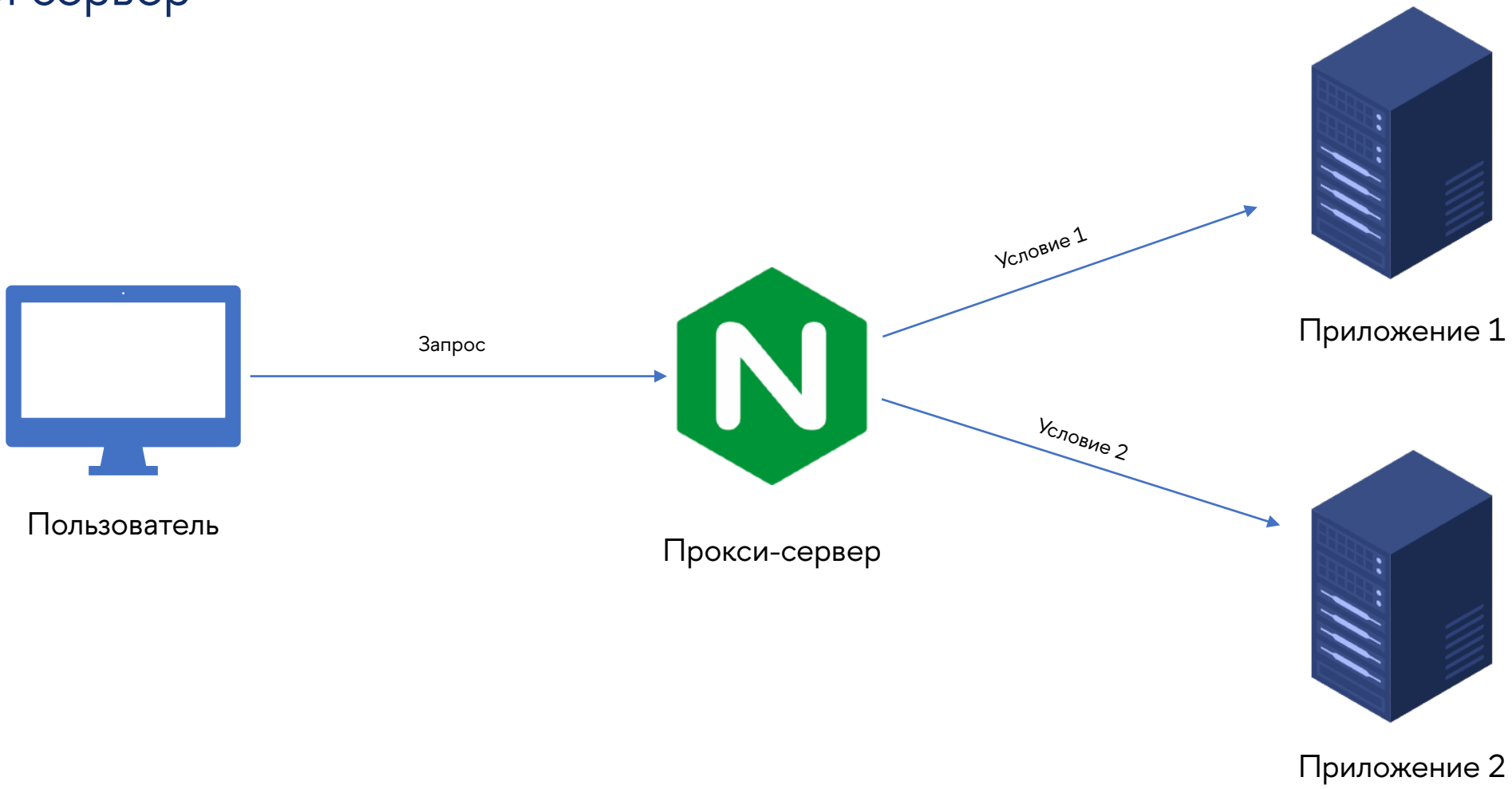


# Прочие важные заголовки

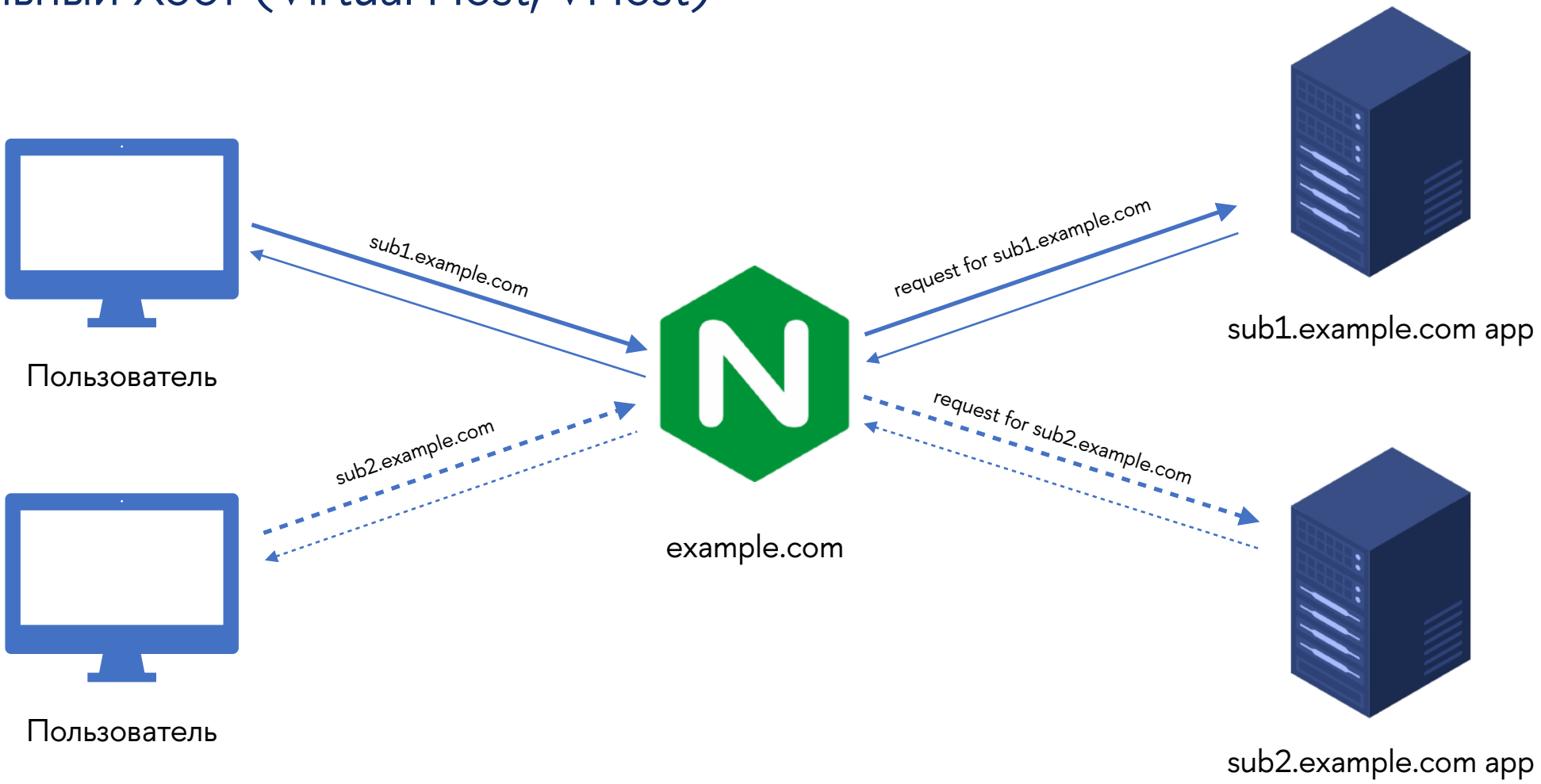
- **Host** - Указывает доменное имя и порт сервера, к которому обращается клиент. (Host: example.com)
- **Content-Type** - Указывает MIME-тип тела запроса. (Content-Type: application/json)
- **Content-Length** - Указывает размер тела HTTP-сообщения в байтах. (Content-Length: 1337)
- **Referer** - Указывает URL предыдущей страницы, с которой пришел пользователь. (Referer: https://example.com/some_path)
- **Origin** - Указывает origin источника запроса (для CORS). (Origin: https://example.com)
- **User-Agent** - Идентифицирует клиентское ПО (браузер, ОС). (User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) ...)
- **X-Forwarded-For (XFF)** - Используется прокси для указания реального IP клиента. (X-Forwarded-For: 192.168.1.1)
- **Transfer-Encoding** - Используется для указания формата отправки тела запроса. (Transfer-Encoding: chunked)
- **Location** - Заголовок ответа. Указывает URL для перенаправления (в ответах 3xx). (Location: https://example.com)
- **Content-Security-Policy (CSP)** - Заголовок ответа. Регулирует правила загрузки скриптов, фреймов и прочего контента (Content-Security-Policy: default-src 'self';)

# Прокси-сервер

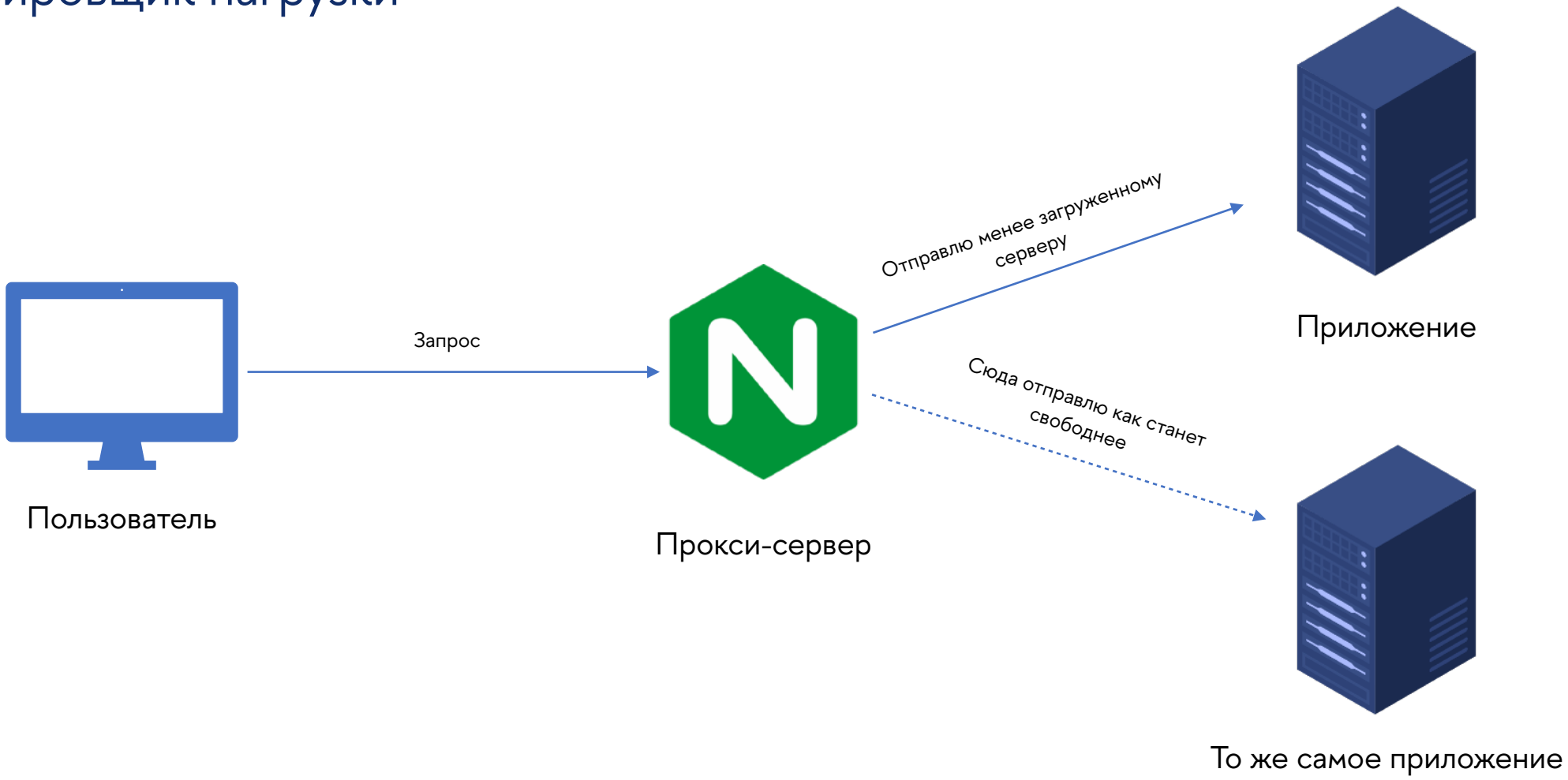
# Прокси сервер



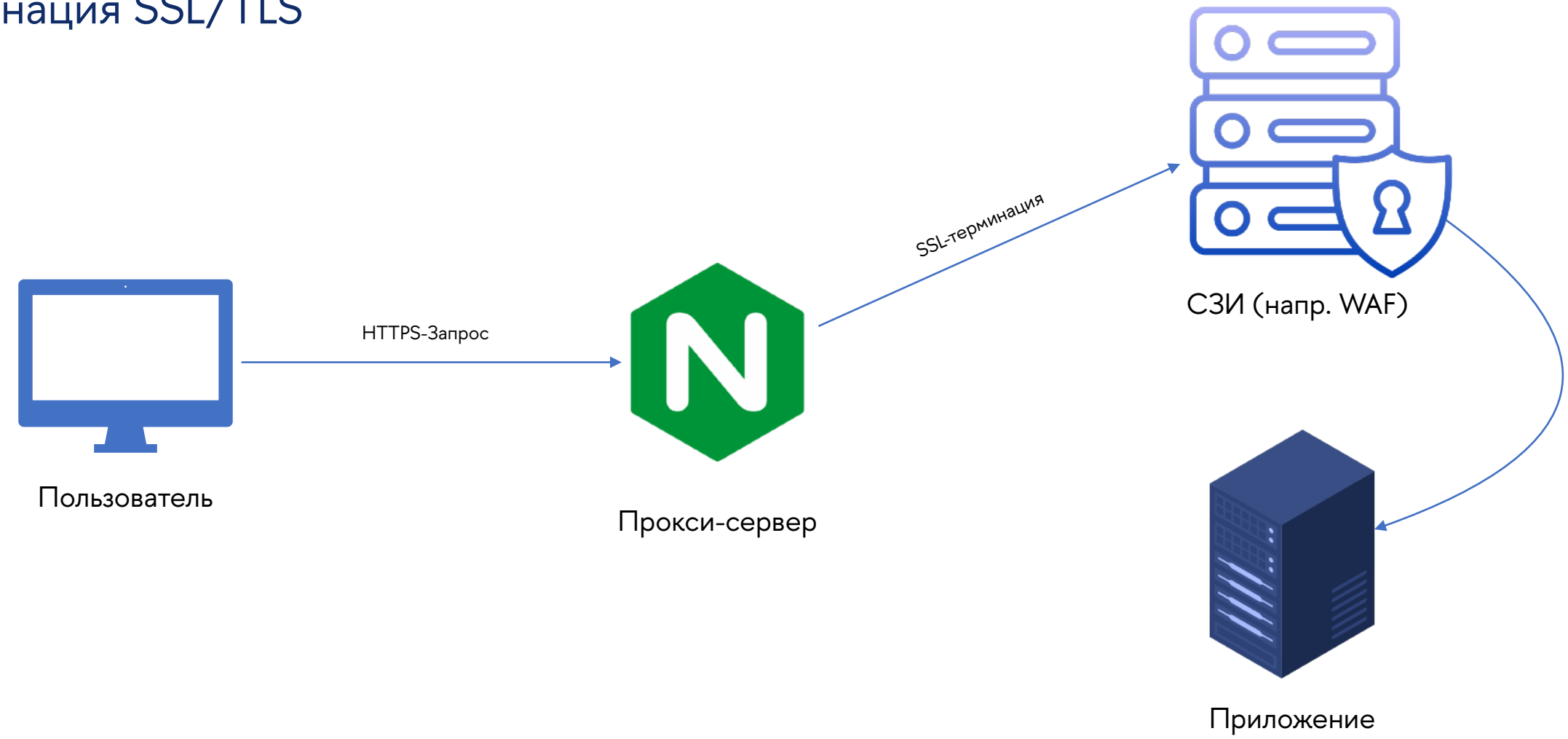
# Виртуальный Хост (Virtual Host, VHost)



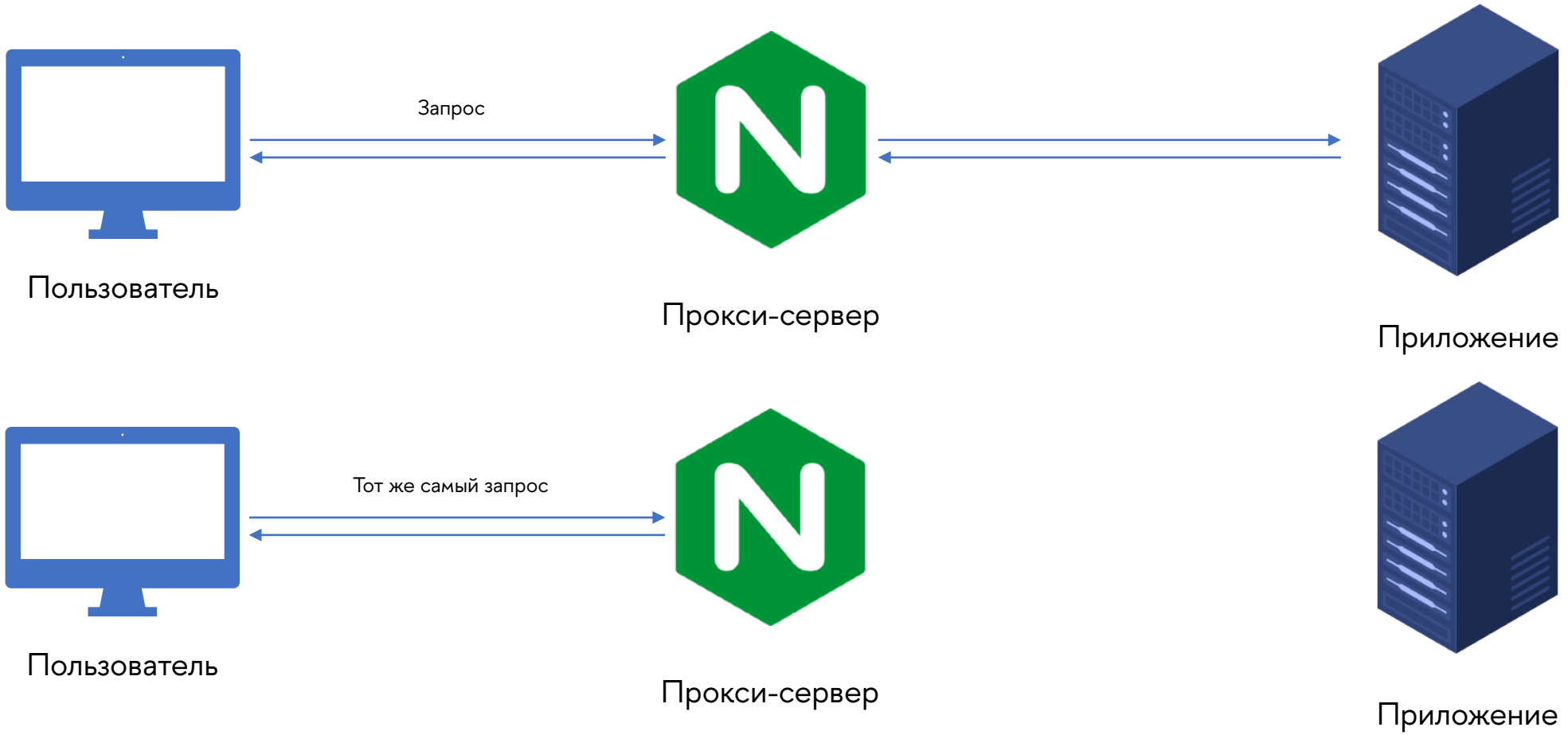
# Балансировщик нагрузки



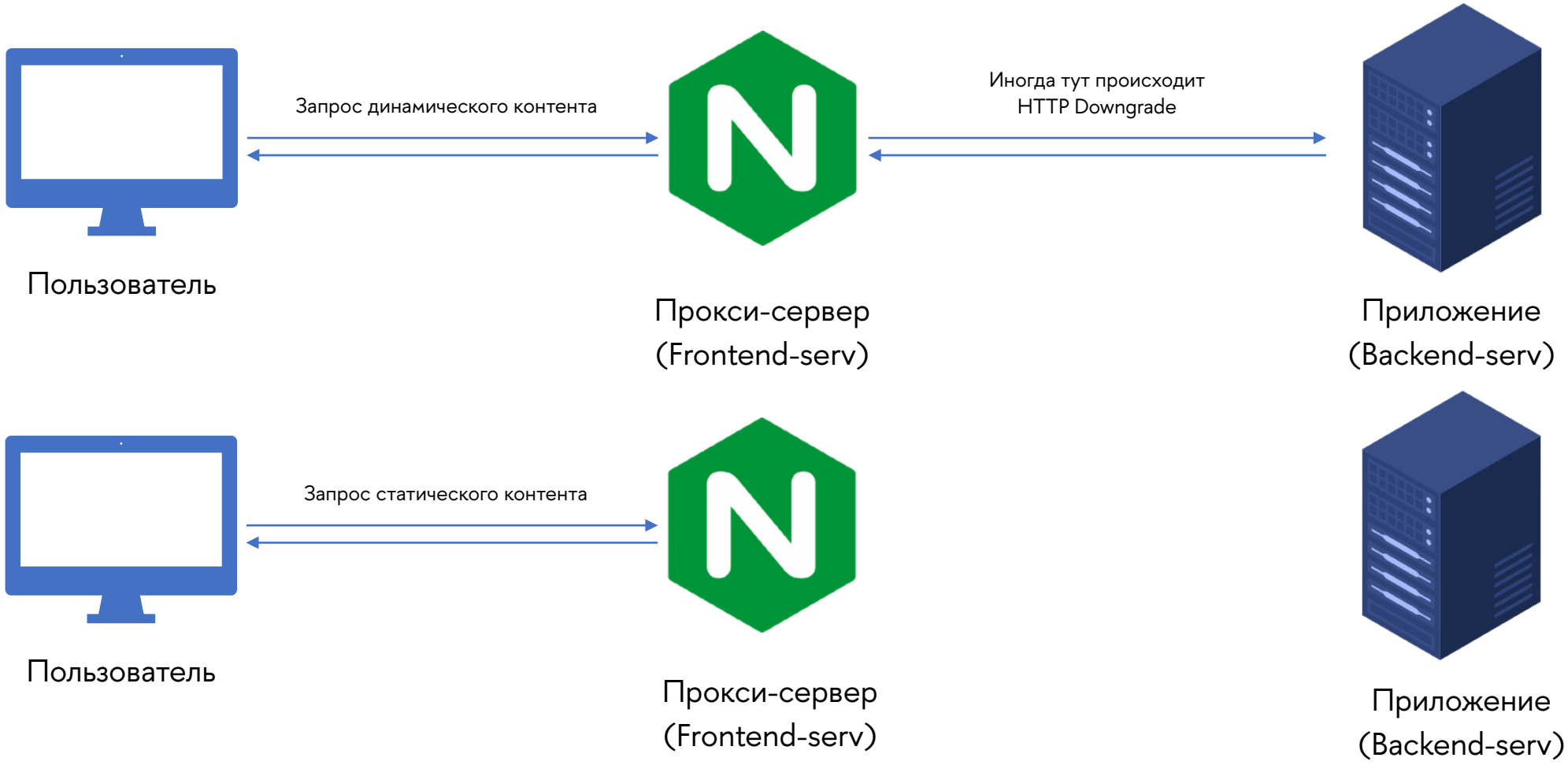
# Терминация SSL/TLS



# Кеширование

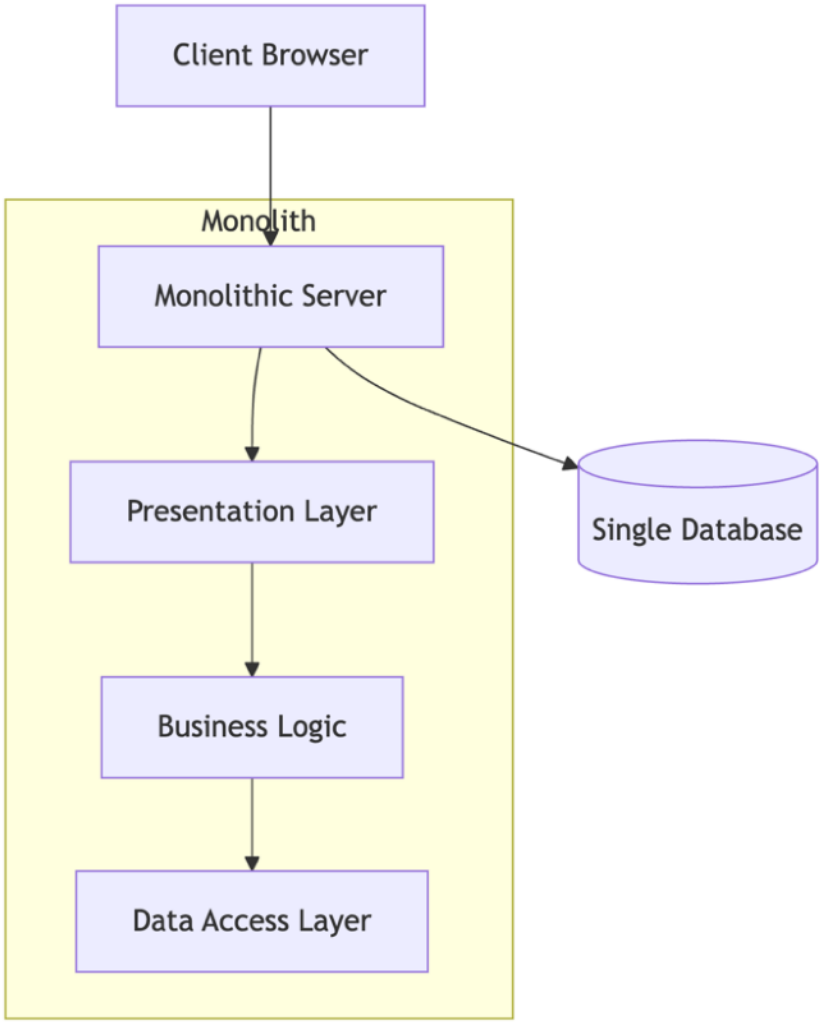


# Frontend- и Backend-сервера

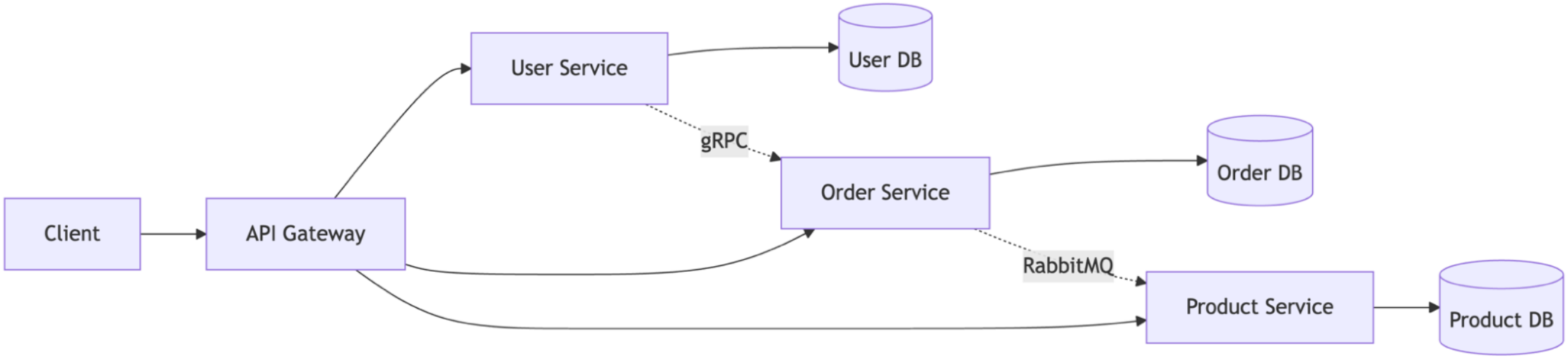


# Архитектуры веб-приложений

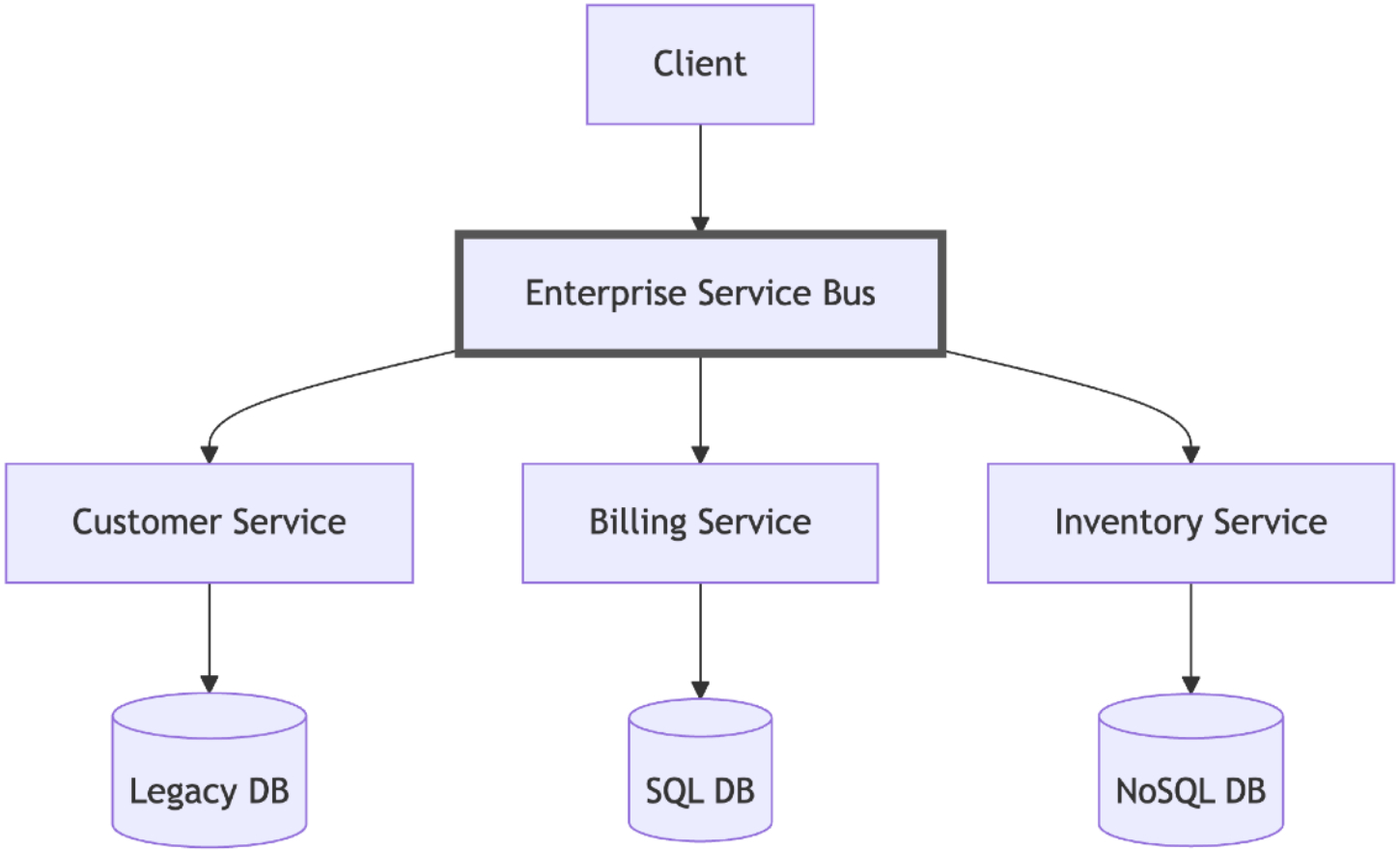
# Монолит


Client Browser
Monolith
Monolithic Server
Presentation Layer
Business Logic
Data Access Layer
Single Database

# Микросервисная архитектура

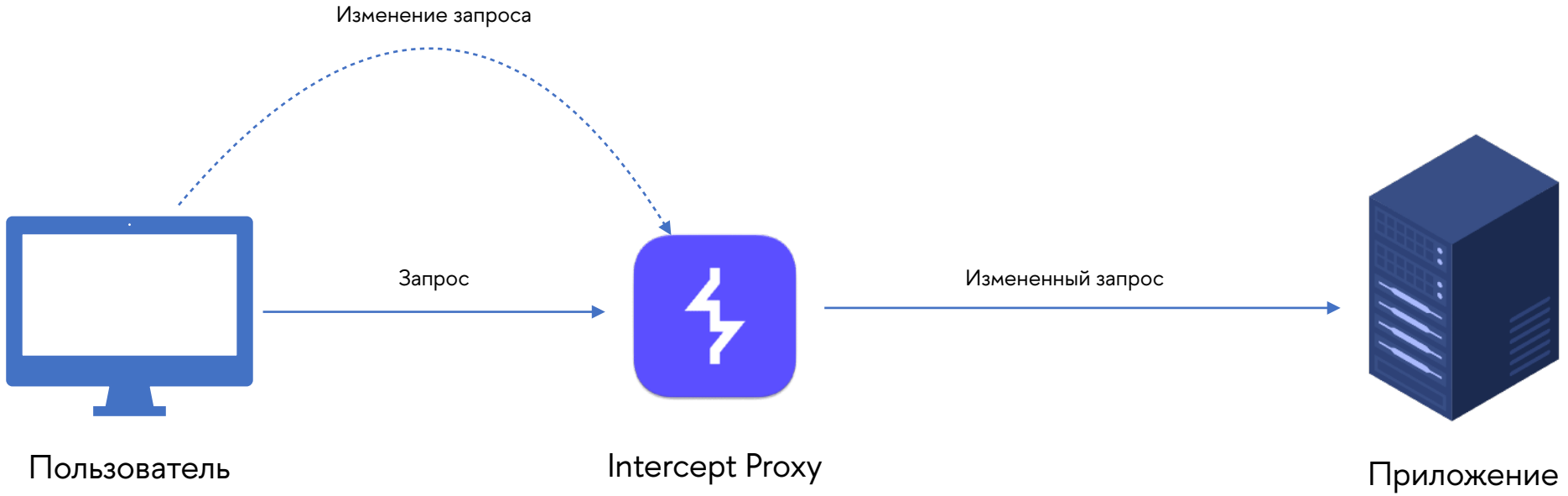


# Сервис-ориентированная архитектура



# Intercept Proxy

# Intercept Proxy



@LEXA_MALOSPAAL

@HUN7_0R_B3_HUN73D


ВЫСШАЯ • ШКОЛА • ЭКОНОМИКИ •